Due: Wed, October 31, beginning of lecture

NOTE: Each problem set only counts 5% of your mark, but it is important to do your own work (but see below). Similar questions will appear on the second term test. You may consult with others concerning the general approach for solving problems on assignments, but you must write up all solutions entirely on your own. Anything else is *plagiarism*, and is subject to the University's Code of Behavior. You will receive 1/5 points for any (non bonus) question/subquestion for which you say "I do not know how to answer this question". You will receive .5/5 points if you just leave the question blankxa. Advice: Do NOT spend an excessive amount of time on any question and especially not on a bonus question. If you wish to spend "free time" thinking about (say) bonus questions that is fine but you should not sacrifice time needed for other courses.

1. Consider the following "roots to coefficients" problem. We are given $\{a_1, \ldots, a_n\}$ and we want to compute the (coefficients of the) polynomial $p(x) = \Pi_{1 \leq i \leq n}(x - a_i) = (x - a_1)(x - a_2) \cdots (x - a_n)$. We will assume the existence of a subroutine for polynomial multiplication and want to derive a divide and conquer algorithm for our roots to coeficients problem. You may assume $n = 2^k$ for some k.

   (a) Assuming the use of a polynomial multiplication subroutine using $O(n^{log_2 3})$ arithmetic operations (e.g. Karatsuba's algorithm), describe a divide and conquer algorithm using $O(n^{log_2 3})$ arithmetic operations for the roots to coeficients problem. Give a recurrence that describes the number of arithmetic operations made by your algorithm and briefly show why this recurrence yields the desired bound.

   [10 points]

   (b) Assuming the use of a polynomial multiplication subroutine using $O(n \log n)$ operations (e.g. using the FFT), what is the arithmetic complexity of your divide and conquer algorithm for the roots to coeficients problem?

   [5 points]

   (c) (Bonus question): Recall the division theorem for polynomials (over a field F). Namely, for polynomials $a(x), b(x)$ there exist unique polynomials $q(x), r(x)$ satisfying $a(x) = b(x) * q(x) + r(x)$ and $degree(r) < degree(b)$. As a consequence, if $p_i \in F$ and $b(x) = (x - p_1)(x - p_2) \ldots (x - p_r)$ then $a(p_i) = r(p_i)$ for $1 \leq i \leq r$.

   Suppose, for some $\alpha > 1$, we can multiply and divide polynomials (i.e. compute the quotient $q(x)$ and remainder $r(x)$ polynomials as above) in $O(n^\alpha)$ arithmetic operations (in the field F).

   Let $a(x)$ be a degree $n-1$ polynomial with $n = 2^k$ for some k. Describe a divide and conquer algorithm (using the division theorem and the roots to coeficients problem) for computing $a(p_1), \ldots, a(p_n)$ using $O(n^\alpha)$ arithmetic operations.

   [10 points]

2. Consider the following "more than a third problem". We are given an array A of $n \geq 2$ elements which cannot be sorted (e.g. the elements are complex numbers) but there is a test for equality of any two elements.

   (a) Describe an $O(n \log n)$ divide and conquer algorithm to find an element (if any) that occurs more than $\lfloor n/3 \rfloor$ times in A.
   Give a recurrence that describes the number of pairwise equality tests made by your algorithm. Derive a bound on the number of equality tests using this recurrence. Hint: How many such elements can A have?
   [10 points]

   (b) For the "more than a third problem" on n inputs, describe a one-sided error randomized algorithm ALG with probability of failure at most $\frac{1}{4}$ and using only $O(n)$ pairwise equality tests. More specifically, if the set A has no element occurring more than $\lfloor n/3 \rfloor$ times, then ALG returns this fact as its output. If A has at least one such element then with probability at least $\frac{3}{4}$, ALG returns one such element. For all inputs, ALG should make at most $c \cdot n$ equality tests for some constant c. Justify the stated error probability. Determine an upper bound for the constant c you are claiming. [10 points]

3. (a) Consider the two machine weighted interval scheduling problem. That is, as in question 1 of the first problem set we want to compute a feasible schedule $\sigma : \{1, \ldots, n\} \rightarrow \{0, 1, 2\}$ so as to maximize profit where now each interval I_j has a profit v_j. Design a $O(n^3)$ time DP algorithm for computing an optimal solution for this problem. What is the complexity (as a function of n) of your algorithm. Provide both semantic and computationally defined arrays and briefly justify the equivalence between these arrrays.
   [10 points]

   (b) (Bonus question) Show how to reduce the complexity of your algorithm to have time $O(n^2)$.
   [5 points]

4. Consider the following additional edit distance costs for the sequence alignment problem (in addition to the matching $\alpha : \Sigma \times \Sigma \rightarrow \Re^{\geq 0}$ and deletion $\delta : \Sigma \rightarrow \Re^{\geq 0}$ costs already discussed in class). Show how to modify the DP algorithm (i.e. how to modify the recursive definition of the desired computational array) to obtain an optimal solution for these modifications. Provide the time complexity of your algorithm.

   (a) There is an adjacent symbol transposition cost $\gamma : \Sigma \times \Sigma \rightarrow \Re^{\geq 0}$. That is, if $(x_i, x_{i+1}) = (a, b) = (y_{j+1}, y_j)$ then we can match x_i with y_j and x_{i+1} with y_{j+1} at a cost of $\gamma(a, b)$.
   [5 points]

   (b) Any sequence S of k consecutive symbols in X or Y can be deleted at a cost of $(k-1) \cdot min_{a \in S} \delta(a) + \cdot max_{a \in S} \delta(a)$.
   [10 points]

5. Consider the following change-making problem: Given an unlimited supply of coins having values $x_1, x_2, \ldots, x_n$ and a value V. The goal is to find (if at all possible) the

smallest set of coins whose total value is exactly V. That is, a feasible solution is a set $m_1, m_2, \ldots m_n$ such that $\sum_k m_k x_k = V$ and the goal is to find a feasible solution so as to minimize $\sum_k m_k$ or to report that no feasible solution exists. [15 points]

6. (a) Let $G = (V, E)$ be a directed acyclic graph (DAG) with $s, t \in V$. Provide a DP algorithm for counting the number of different directed paths from s to t. (Two paths are different if there is at least one node in one path that is not in the other.) [10 points]

   (b) Discuss whether or not your algorithm can be extended so as to count the number of simple s to t paths in a directed (but not necessarily simple) graph. [5 points]

7. Perhaps more to follow